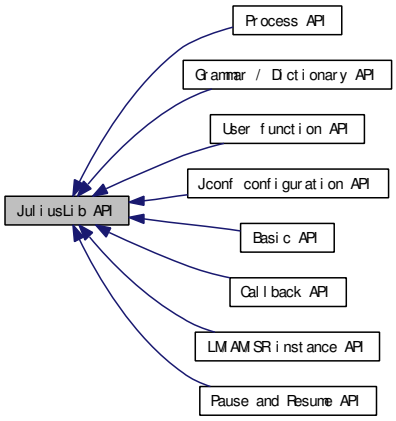
# Chapter 6

# Julius Module Documentation

## 6.1 JuliusLib API

Collaboration diagram for JuliusLib API:




Here is a reference of all Julius library API functions.

### Modules

- Basic API

  *Basic functions to start-up and initialize engines.*

- Callback API

  *Functions to add callback to get results and status.*

- Pause and Resume API

  *Functions to pause / resume engine inputs.*

- User function API

  *Functions to register user function to be applied inside Julius.*

- Process API

  *Functions to create / remove / (de)activate recognition process and models on live.*

- Grammar / Dictionary API

  *Functions to manage grammars or word dictionaries at run time.*

- Jconf configuration API

  *Functions to load / create configuration parameters.*

- LM/AM/SR instance API

  *Functions to handle modules and processes directly.*

### 6.1.1 Detailed Description

Here is a reference of all Julius library API functions.

## 6.2 Basic API

Collaboration diagram for Basic API:


Basic functions to start-up and initialize engines.

### Functions

- void j_enable_debug_message ()

  *Enable debug messages in JuliusLib to log.*

- void j_disable_debug_message ()

  *Disable debug messages in JuliusLib to log.*

- void j_enable_verbose_message ()

  *Enable verbose messages in JuliusLib to log.*

- void j_disable_verbose_message ()

  *Disable verbose messages in JuliusLib to log.*

- boolean j_adin_init (Recog *recog)

  *Initialize and setup A/D-in device specified by the configuration for recognition.*

- boolean j_adin_init_user (Recog *recog, void *arg)

  *Initialize function when using user-defined A/D-in function.*

- char * j_get_current_filename ()

  *Return current input speech file name.*

- void j_recog_info (Recog *recog)

  *Output all configurations and system informations into log.*

- void j_output_argument_help (FILE *fp)

  *Output help document.*

- int j_open_stream (Recog *recog, char *file_or_dev_name)

  *Open input stream.*

- int j_recognize_stream (Recog *recog)

  *Recognize an input stream.*

- boolean j_add_option (char *fmt, int argnum, int reqargnum, char *desc, boolean(*func)(Jconf *jconf, char *arg[ ], int argnum))

  *Add a user-defined option to Julius.*

### 6.2.1 Detailed Description

Basic functions to start-up and initialize engines.

### 6.2.2 Function Documentation

#### 6.2.2.1 boolean j_adin_init (Recog * *recog*)

Initialize and setup A/D-in device specified by the configuration for recognition.

When threading is enabled for the device, A/D-in thread will start inside this function.

**Parameters:**

***recog*** [in] engine instance

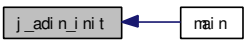
**Returns:**

TRUE on success, FALSE on failure.

Definition at line 521 of file jfunc.c.

Referenced by main().

Here is the caller graph for this function:




#### 6.2.2.2 boolean j_adin_init_user (Recog * *recog*, void * *arg*)

Initialize function when using user-defined A/D-in function.

Almost the same with j_adin_init() except that it does not initialize an input device.

**Parameters:**

***recog*** [in] engine instance

***arg*** [in] argument

**Returns:**

TRUE on success, else FALSE on error.

Definition at line 565 of file jfunc.c.

#### 6.2.2.3 char* j_get_current_filename ()

Return current input speech file name.

Invalid when MFCC input.

**Returns:**

the file name.

Definition at line 602 of file jfunc.c.

Referenced by main_recognition_stream_loop().

Here is the caller graph for this function:


#### 6.2.2.4 void j_recog_info (Recog * *recog*)

Output all configurations and system informations into log.

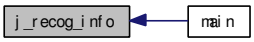
**Parameters:**

***recog*** [in] engine instance

Definition at line 626 of file jfunc.c.

Referenced by main().

Here is the caller graph for this function:




#### 6.2.2.5 void j_output_argument_help (FILE * *fp*)

Output help document.

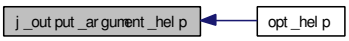
**Parameters:**

***fp*** [in] file pointer to output help

Definition at line 45 of file m_usage.c.

Referenced by opt_help().

Here is the caller graph for this function:




#### 6.2.2.6 int j_open_stream (Recog * *recog*, char * *file_or_dev_name*)

Open input stream.

**Parameters:**

***recog*** [i/o] engine instance

***file_or_dev_name*** [in] file or device name of the device

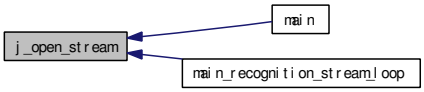
**Returns:**

0 on success, -1 on error, -2 on decice initializatino error.

Definition at line 447 of file recogmain.c.

Referenced by main(), and main_recognition_stream_loop().

Here is the caller graph for this function:




#### 6.2.2.7 int j_recognize_stream (Recog * *recog*)

Recognize an input stream.

This function repeat recognition process for the whole input stream, using segmentation and detection if required. It ends when the whole input has been processed.

When a recognition stop is requested from application, the following callbacks will be called in turn: CALLBACK_EVENT_PAUSE, CALLBACK_PAUSE_FUNCTION, CALLBACK_EVENT_RESUME. After finishing executing all functions in these callbacks, recognition will restart. If you have something to be processed while recognition stops, you should write the function as callback to CALLBACK_PAUSE_-FUNCTION. Note that recognition will restart immediately after all functions registered in CALLBACK_-PAUSE_FUNCTION has been finished.

**Parameters:**

***recog*** [i/o] engine instance

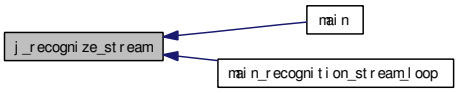
**Returns:**

0 when finished recognizing all the input stream to the end, or -1 on error.

Definition at line 1216 of file recogmain.c.

Referenced by main(), and main_recognition_stream_loop().

Here is the caller graph for this function:




#### 6.2.2.8 boolean j_add_option (char * *fmt*, int *argnum*, int *reqargnum*, char * *desc*, boolean(*)(Jconf *jconf, char *arg[ ], int argnum) *func*)

Add a user-defined option to Julius.

When reqargnum is lower than argnum, the first (reqargnum) arguments are required and the rest (argnum - reqargnum) options are optional.

**Parameters:**

***fmt*** [in] option string (should begin with '-')

***argnum*** [in] total number of argument for this option (including optional)

***reqargnum*** [in] number of required argument

***desc*** [in] description string for help

***func*** [in] option handling function

**Returns:**

TRUE on success, FALSE on failure

Definition at line 129 of file useropt.c.

## 6.3 Callback API

Collaboration diagram for Callback API:


Functions to add callback to get results and status.

### Functions

- int callback_add (Recog *recog, int code, void(*func)(Recog *recog, void *data), void *data)

  *Register a function to a callback registory.*

- int callback_add_adin (Recog *recog, int code, void(*func)(Recog *recog, SP16 *buf, int len, void *data), void *data)

  *Register a function to the A/D-in type callback registory.*

- boolean callback_exist (Recog *recog, int code)

  *Check if at least one function has been registered to a callback repository.*

- boolean callback_delete (Recog *recog, int id)

  *Delete an already registered function from callback.*

### 6.3.1 Detailed Description

Functions to add callback to get results and status.

### 6.3.2 Function Documentation

#### 6.3.2.1 int callback_add (Recog * *recog*, int *code*, void(*)(Recog *recog, void *data) *func*, void * *data*)

Register a function to a callback registory.

**Parameters:**

***recog*** [i/o] engine instance

***code*** [in] code in which the function will be registered

***func*** [in] function

***data*** [in] user-specified argument to be passed when the function is called inside Julius

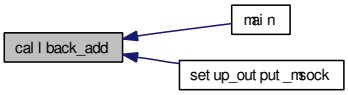
**Returns:**

global callback ID unique for the whole process, or -1 on error.

Definition at line 135 of file callback.c.

Referenced by main(), and setup_output_msock().

Here is the caller graph for this function:




#### 6.3.2.2 int callback_add_adin (Recog * *recog*, int *code*, void(*)(Recog *recog, SP16 *buf, int len, void *data) *func*, void * *data*)

Register a function to the A/D-in type callback registory.

**Parameters:**

***recog*** [i/o] engine instance

***code*** [in] code in which the function will be registered

***func*** [in] function

***data*** [in] user-specified argument to be passed when the function is called inside Julius

**Returns:**

global callback ID unique for the whole process, or -1 on error.

Definition at line 161 of file callback.c.

#### 6.3.2.3 boolean callback_exist (Recog * *recog*, int *code*)

Check if at least one function has been registered to a callback repository.

**Parameters:**

***recog*** [in] engine instance

***code*** [in] callback code

**Returns:**

TRUE when at least one is registered, or FALSE if none.

Definition at line 246 of file callback.c.

#### 6.3.2.4 boolean callback_delete (Recog * *recog*, int *id*)

Delete an already registered function from callback.

**Parameters:**

***recog*** [i/o] engine instance

***id*** [in] global callback ID to delete

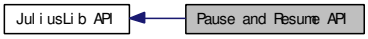
**Returns:**

TRUE on success, or FALSE on failure.

Definition at line 271 of file callback.c.

## 6.4 Pause and Resume API

Collaboration diagram for Pause and Resume API:




Functions to pause / resume engine inputs.

### Functions

- void j_request_pause (Recog *recog)
  *Request engine to stop recognition.*
- void j_request_terminate (Recog *recog)
  *Request engine to terminate recognition immediately.*
- void j_request_resume (Recog *recog)
  *Resume the engine which has already paused or terminated.*

### 6.4.1 Detailed Description

Functions to pause / resume engine inputs.

### 6.4.2 Function Documentation

#### 6.4.2.1 void j_request_pause (Recog * *recog*)

Request engine to stop recognition.

If the engine is recognizing a speech input, it will stop after the current recognition ended.

**Parameters:**

***recog*** [in] engine instance

Definition at line 52 of file jfunc.c.

#### 6.4.2.2 void j_request_terminate (Recog * *recog*)

Request engine to terminate recognition immediately.

Even if the engine is recognizing a speech input, it will stop immediately (in this case the current input will be lost).

**Parameters:**

***recog*** [in] engine instance

Definition at line 85 of file jfunc.c.

#### 6.4.2.3 void j_request_resume (Recog * *recog*)

Resume the engine which has already paused or terminated.

**Parameters:**

***recog***

Definition at line 117 of file jfunc.c.

## 6.5 User function API

Collaboration diagram for User function API:


Functions to register user function to be applied inside Julius.

### Functions

- boolean j_regist_user_lm_func (PROCESS_LM *lm, LOGPROB(*unifunc)(WORD_INFO *winfo, WORD_ID w, LOGPROB ngram_prob), LOGPROB(*bifunc)(WORD_INFO *winfo, WORD_ID context, WORD_ID w, LOGPROB ngram_prob), LOGPROB(*probfunc)(WORD_INFO *winfo, WORD_ID *contexts, int context_len, WORD_ID w, LOGPROB ngram_prob))

  *Assign user-defined language scoring functions into a LM processing instance.*

- boolean j_regist_user_param_func (Recog *recog, boolean(*user_calc_vector)(MFCCCalc *, SP16 *, int))

  *Assign a user-defined parameter extraction function to engine instance.*

### 6.5.1 Detailed Description

Functions to register user function to be applied inside Julius.

### 6.5.2 Function Documentation

#### 6.5.2.1 boolean j_regist_user_lm_func (PROCESS_LM * *lm*, LOGPROB(*)(WORD_INFO *winfo, WORD_ID w, LOGPROB ngram_prob) *unifunc*, LOGPROB(*)(WORD_INFO *winfo, WORD_ID context, WORD_ID w, LOGPROB ngram_prob) *bifunc*, LOGPROB(*)(WORD_INFO *winfo, WORD_ID *contexts, int context_len, WORD_ID w, LOGPROB ngram_prob) *probfunc*)

Assign user-defined language scoring functions into a LM processing instance.

This should be called after engine instance creation and before j_final_fusion() is called. Remember that you should also specify "-userlm" option at jconf to use user-define language scoring.

**Parameters:**

***lm*** [i/o] LM processing instance

***unifunc*** [in] pointer to the user-defined unigram function

***bifunc*** [in] pointer to the user-defined bi-igram function

***probfunc*** [in] pointer to the user-defined N-gram function

**Returns:**

TRUE on success, FALSE on failure.

Definition at line 719 of file jfunc.c.

#### 6.5.2.2 boolean j_regist_user_param_func (Recog * *recog*, boolean(*)(MFCCCalc *, SP16 *, int) *user_calc_vector*)

Assign a user-defined parameter extraction function to engine instance.

**Parameters:**

***recog*** [i/o] engine instance

***user_calc_vector*** [in] pointer to function of parameter extraction

**Returns:**

TRUE on success, FALSE on error.

Definition at line 748 of file jfunc.c.

## 6.6 Process API

Collaboration diagram for Process API:


Functions to create / remove / (de)activate recognition process and models on live.

### Functions

- boolean j_process_deactivate (Recog *recog, char *name)

  *De-activate a recognition process instance designated by its name.*

- boolean j_process_deactivate_by_id (Recog *recog, int id)

  *De-activate a recognition process instance designated by its ID.*

- boolean j_process_activate (Recog *recog, char *name)

  *Activate a recognition process instance that has been made inactive, by its name.*

- boolean j_process_activate_by_id (Recog *recog, int id)

  *Activate a recognition process instance that has been made inactive, by the ID.*

- boolean j_process_add_lm (Recog *recog, JCONF_LM *lmconf, JCONF_SEARCH *sconf, char *name)

  *Create a new recognizer with a new LM and SR configurations.*

- boolean j_process_remove (Recog *recog, JCONF_SEARCH *sconf)

  *Remove a recognition process instance.*

- boolean j_process_lm_remove (Recog *recog, JCONF_LM *lmconf)

  *Remove an LM process instance.*

- boolean j_process_am_remove (Recog *recog, JCONF_AM *amconf)

  *Remove an AM process instance (experimental).*

### 6.6.1 Detailed Description

Functions to create / remove / (de)activate recognition process and models on live.

### 6.6.2 Function Documentation

#### 6.6.2.1 boolean j_process_deactivate (Recog * *recog*, char * *name*)

De-activate a recognition process instance designated by its name.

The process will actually pauses at the next recognition interval.

**Parameters:**

***recog*** [i/o] engine instance

***name*** [in] SR name to deactivate

**Returns:**

TRUE on success, or FALSE on failure.

Definition at line 1010 of file jfunc.c.

#### 6.6.2.2 boolean j_process_deactivate_by_id (Recog * *recog*, int *id*)

De-activate a recognition process instance designated by its ID.

The process will actually pauses at the next recognition interval.

**Parameters:**

***recog*** [i/o] engine instance

***id*** [in] SR ID to deactivate

**Returns:**

TRUE on success, or FALSE on failure.

Definition at line 1054 of file jfunc.c.

#### 6.6.2.3 boolean j_process_activate (Recog * *recog*, char * *name*)

Activate a recognition process instance that has been made inactive, by its name.

The process will actually starts at the next recognition interval.

**Parameters:**

***recog*** [i/o] engine instance

***name*** [in] SR name to activate

**Returns:**

TRUE on success, or FALSE on failure.

Definition at line 1098 of file jfunc.c.

#### 6.6.2.4 boolean j_process_activate_by_id (Recog * *recog*, int *id*)

Activate a recognition process instance that has been made inactive, by the ID.

The process will actually starts at the next recognition interval.

**Parameters:**

***recog*** [i/o] engine instance

***id*** [in] SR ID to activate

**Returns:**

TRUE on success, or FALSE on failure.

Definition at line 1142 of file jfunc.c.

#### 6.6.2.5 boolean j_process_add_lm (Recog * *recog*, JCONF_LM * *lmconf*, JCONF_SEARCH * *sconf*, char * *name*)

Create a new recognizer with a new LM and SR configurations.

This function creates new LM process instance and recognition process instance corresponding to the given LM and SR configurations. AM process to be assigned to them is the current default AM. Both the new LM and SR will be assigned the same instance name.

**Parameters:**

***recog*** [i/o] engine instance

***lmconf*** [in] a new LM configuration

***sconf*** [in] a new SR configuration

***name*** [in] name of the new instances

**Returns:**

TRUE on success, FALSE on error.

Definition at line 1195 of file jfunc.c.

#### 6.6.2.6 boolean j_process_remove (Recog * *recog*, JCONF_SEARCH * *sconf*)

Remove a recognition process instance.

The specified search conf will also be released and destroyed inside this function.

**Parameters:**

***recog*** [in] engine instance

***sconf*** [in] SEARCH configuration corresponding to the target recognition process to remove

**Returns:**

TRUE on success, or FALSE on failure.

Definition at line 1258 of file jfunc.c.

#### 6.6.2.7 boolean j_process_lm_remove (Recog * *recog*, JCONF_LM * *lmconf*)

Remove an LM process instance.

The specified lm conf will also be released and destroyed inside this function.

**Parameters:**

***recog*** [in] engine instance

***lmconf*** [in] LM configuration corresponding to the target LM process to remove

**Returns:**

TRUE on success, or FALSE on failure.

Definition at line 1336 of file jfunc.c.

#### 6.6.2.8 boolean j_process_am_remove (Recog * *recog*, JCONF_AM * *amconf*)

Remove an AM process instance (experimental).

The specified am conf will also be released and destroyed inside this function.

**Parameters:**

***recog*** [in] engine instance

***amconf*** [in] AM configuration corresponding to the target AM process to remove

**Returns:**

TRUE on success, or FALSE on failure.

Definition at line 1424 of file jfunc.c.

## 6.7 Grammar / Dictionary API

Collaboration diagram for Grammar / Dictionary API:


Functions to manage grammars or word dictionaries at run time.

### Functions

- void multigram_add_gramlist (char *dfafile, char *dictfile, JCONF_LM *j, int lmvar)
  *Add a grammar to the grammar list to be read at startup.*
- void multigram_remove_gramlist (JCONF_LM *j)
  *Remove the grammar list to be read at startup.*
- boolean multigram_add_prefix_list (char *prefix_list, char *cwd, JCONF_LM *j, int lmvar)
  *Add multiple grammars given by their prefixs to the grammar list.*
- boolean multigram_add_prefix_filelist (char *listfile, JCONF_LM *j, int lmvar)
  *Add multiple grammars from prefix list file to the grammar list.*
- void schedule_grammar_update (Recog *recog)
  *Request engine to check update of all grammar and re-construct the glocal lexicon if needed.*
- void j_reset_reload (Recog *recog)
  *Clear the grammar re-construction flag.*
- boolean multigram_build (RecogProcess *r)
  *Check for global grammar and (re-)build tree lexicon if needed.*
- void multigram_add (DFA_INFO *dfa, WORD_INFO *winfo, char *name, PROCESS_LM *lm)
  *Add a new grammar to the current list of grammars.*
- boolean multigram_delete (int delid, PROCESS_LM *lm)
  *Mark a grammar in the grammar list to be deleted at the next grammar update.*
- void multigram_delete_all (PROCESS_LM *lm)
  *Mark all grammars to be deleted at next grammar update.*
- int multigram_activate (int gid, PROCESS_LM *lm)
  *Activate a grammar in the grammar list.*
- int multigram_deactivate (int gid, PROCESS_LM *lm)
  *Deactivate a grammar in the grammar list.*
- boolean multigram_update (PROCESS_LM *lm)
  *Update global grammar if needed.*

- int multigram_get_all_num (PROCESS_LM *lm)

  *Get the number of current grammars (both active and inactive).*

- int multigram_get_gram_from_category (int category, PROCESS_LM *lm)

  *Get which grammar the given category belongs to.*

- MULTIGRAM * multigram_get_grammar_by_name (PROCESS_LM *lm, char *gramname)

  *Find a grammar in LM by its name.*

- MULTIGRAM * multigram_get_grammar_by_id (PROCESS_LM *lm, unsigned short id)

  *Find a grammar in LM by its ID number.*

- boolean multigram_add_words_to_grammar (PROCESS_LM *lm, MULTIGRAM *m, WORD_INFO *winfo)

  *Append words to a grammar.*

- boolean multigram_add_words_to_grammar_by_name (PROCESS_LM *lm, char *gramname, WORD_INFO *winfo)

  *Append words to a grammar, given by its name.*

- boolean multigram_add_words_to_grammar_by_id (PROCESS_LM *lm, unsigned short id, WORD_INFO *winfo)

  *Append words to a grammar, given by its ID number.*

### 6.7.1 Detailed Description

Functions to manage grammars or word dictionaries at run time.

### 6.7.2 Function Documentation

#### 6.7.2.1 void multigram_add_gramlist (char * *dfafile*, char * *dictfile*, JCONF_LM * *j*, int *lmvar*)

Add a grammar to the grammar list to be read at startup.

**Parameters:**

***dfafile*** [in] DFA file

***dictfile*** [in] dictionary file

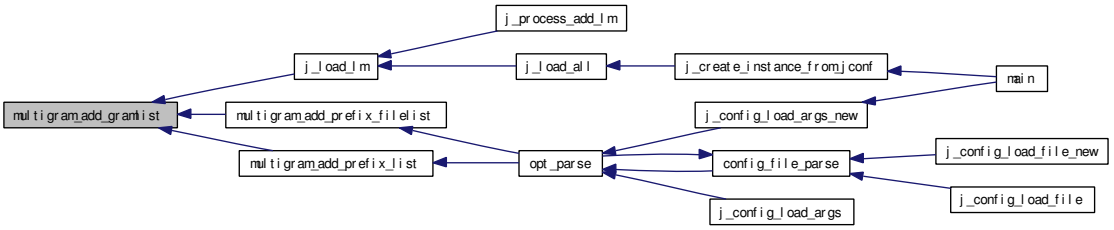
***j*** [in] LM configuration variables

***lmvar*** [in] LM type variant id

Definition at line 66 of file gramlist.c.

Referenced by j_load_lm(), multigram_add_prefix_filelist(), and multigram_add_prefix_list().

Here is the caller graph for this function:




#### 6.7.2.2 void multigram_remove_gramlist (JCONF_LM * *j*)

Remove the grammar list to be read at startup.

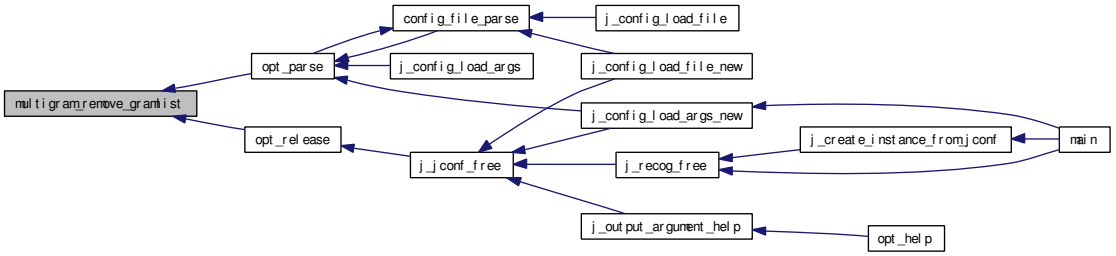
**Parameters:**

***j*** [in] LM configuration variables

Definition at line 103 of file gramlist.c.

Referenced by opt_parse(), and opt_release().

Here is the caller graph for this function:




#### 6.7.2.3 boolean multigram_add_prefix_list (char * *prefix_list*, char * *cwd*, JCONF_LM * *j*, int *lmvar*)

Add multiple grammars given by their prefixs to the grammar list.

This function read in several grammars, given a prefix string that contains a list of file prefixes separated by comma: "foo" or "foo,bar". For each prefix, string ".dfa" and ".dict" will be appended to read dfa file and dict file. The read grammars will be added to the grammar list.

**Parameters:**

***prefix_list*** [in] string that contains comma-separated list of grammar path prefixes

***cwd*** [in] string of current working directory

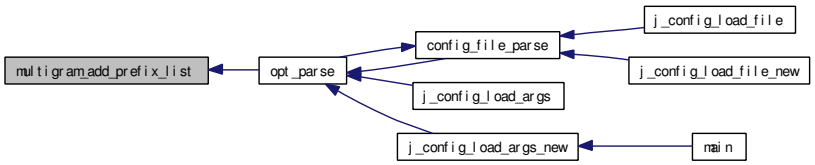
***j*** [in] LM configuration variables

***lmvar*** [in] LM type variant id

Definition at line 163 of file gramlist.c.

Referenced by opt_parse().

Here is the caller graph for this function:




#### 6.7.2.4 boolean multigram_add_prefix_filelist (char * *listfile*, JCONF_LM * *j*, int *lmvar*)

Add multiple grammars from prefix list file to the grammar list.

This function read in multiple grammars at once, given a file that contains a list of grammar prefixes, each per line.

For each prefix, string ".dfa" and ".dict" will be appended to read the corresponding dfa and dict file. The read grammars will be added to the grammar list.

**Parameters:**

***listfile*** [in] path of the prefix list file

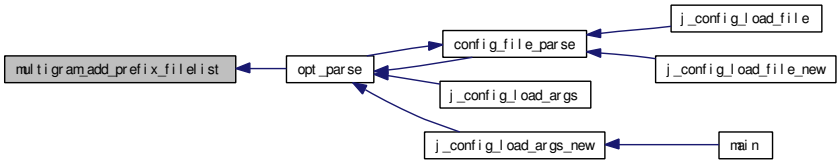
***j*** [in] LM configuration variables

***lmvar*** [in] LM type variant id

Definition at line 265 of file gramlist.c.

Referenced by opt_parse().

Here is the caller graph for this function:




#### 6.7.2.5 void schedule_grammar_update (Recog * *recog*)

Request engine to check update of all grammar and re-construct the glocal lexicon if needed.

The actual update will be done between input segment. This function should be called after some grammars are modified.

**Parameters:**

***recog*** [in] engine instance

Definition at line 152 of file jfunc.c.

#### 6.7.2.6 void j_reset_reload (Recog * *recog*)

Clear the grammar re-construction flag.

**Parameters:**

***recog*** [in] engine instance

Definition at line 197 of file jfunc.c.

#### 6.7.2.7 boolean multigram_build (RecogProcess * *r*)

Check for global grammar and (re-)build tree lexicon if needed.

If any modification of the global grammar has been occured, the tree lexicons and some other data for recognition will be re-constructed from the updated global grammar.

**Parameters:**

***r*** [in] recognition process instance

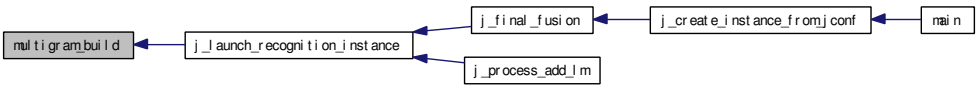
**Returns:**

TRUE on success, FALSE on error.

Definition at line 173 of file multi-gram.c.

Referenced by j_launch_recognition_instance().

Here is the caller graph for this function:




#### 6.7.2.8 void multigram_add (DFA_INFO * *dfa*, WORD_INFO * *winfo*, char * *name*, PROCESS_LM * *lm*)

Add a new grammar to the current list of grammars.

The list of grammars which the LM instance keeps currently is at lm->grammars. The new grammar is flaged at "newbie" and "active", to be treated properly at the next grammar update check.

**Parameters:**

***dfa*** [in] DFA information of the new grammar.

***winfo*** [in] dictionary information of the new grammar.

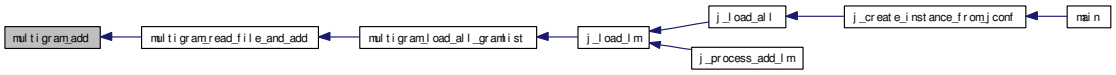
***name*** [in] name string of the new grammar.

***lm*** [i/o] LM processing instance

Definition at line 273 of file multi-gram.c.

Referenced by multigram_read_file_and_add().

Here is the caller graph for this function:




#### 6.7.2.9 boolean multigram_delete (int *delid*, PROCESS_LM * *lm*)

Mark a grammar in the grammar list to be deleted at the next grammar update.

**Parameters:**

***delid*** [in] grammar id to be deleted

***lm*** [i/o] LM processing instance

**Returns:**

TRUE on normal exit, or FALSE if the specified grammar is not found in the grammar list.

Definition at line 326 of file multi-gram.c.

#### 6.7.2.10 void multigram_delete_all (PROCESS_LM * *lm*)

Mark all grammars to be deleted at next grammar update.

**Parameters:**

***lm*** [i/o] LM processing instance

Definition at line 359 of file multi-gram.c.

#### 6.7.2.11 int multigram_activate (int *gid*, PROCESS_LM * *lm*)

Activate a grammar in the grammar list.

The specified grammar will only be marked as to be activated in the next grammar update timing.

**Parameters:**

***gid*** [in] grammar ID to be activated

***lm*** [i/o] LM processing instance

**Returns:**

0 on success, -1 on error (when specified grammar not found), of 1 if already active

Definition at line 441 of file multi-gram.c.

#### 6.7.2.12 int multigram_deactivate (int *gid*, PROCESS_LM * *lm*)

Deactivate a grammar in the grammar list.

The words of the de-activated grammar will not be expanded in the recognition process. This feature enables rapid switching of grammars without re-building tree lexicon. The de-activated grammar will again be activated by calling multigram_activate().

**Parameters:**

***gid*** [in] grammar ID to be de-activated

***lm*** [i/o] LM processing instance

**Returns:**

0 on success, -1 on error (when specified grammar not found), of 1 if already inactive

Definition at line 503 of file multi-gram.c.

#### 6.7.2.13 boolean multigram_update (PROCESS_LM * *lm*)

Update global grammar if needed.

This function checks for any modification in the grammar list from previous call, and update the global grammar if needed.

If there are grammars marked to be deleted in the grammar list, they will be actually deleted from memory. Then the global grammar is built from scratch using the rest grammars. If there are new grammars, they are appended to the current global grammar.

**Parameters:**

***lm*** [i/o] LM processing instance

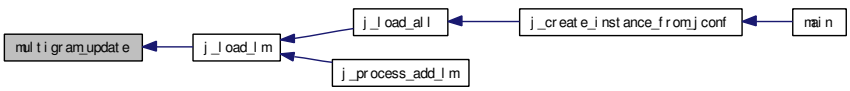
**Returns:**

TRUE when any of add/delete/active/inactive occurs, or FALSE if nothing modified.

Definition at line 615 of file multi-gram.c.

Referenced by j_load_lm().

Here is the caller graph for this function:




#### 6.7.2.14 int multigram_get_all_num (PROCESS_LM * *lm*)

Get the number of current grammars (both active and inactive).

**Parameters:**

***lm*** [i/o] LM processing instance

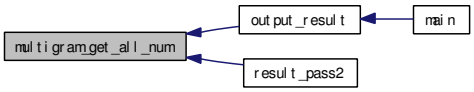
**Returns:**

the number of grammars.

Definition at line 923 of file multi-gram.c.

Referenced by output_result(), and result_pass2().

Here is the caller graph for this function:




#### 6.7.2.15 int multigram_get_gram_from_category (int *category*, PROCESS_LM * *lm*)

Get which grammar the given category belongs to.

**Parameters:**

***category*** word category ID

***lm*** [i/o] LM processing instance

**Returns:**

the id of the belonging grammar.

Definition at line 955 of file multi-gram.c.

#### 6.7.2.16 MULTIGRAM* multigram_get_grammar_by_name (PROCESS_LM * *lm*, char * *gramname*)

Find a grammar in LM by its name.

**Parameters:**

***lm*** [in] LM process instance

***gramname*** [in] grammar name

**Returns:**

poitner to the grammar, or NULL if not found.

Definition at line 1019 of file multi-gram.c.

Referenced by multigram_add_words_to_grammar_by_name().

Here is the caller graph for this function:

#### 6.7.2.17 MULTIGRAM* multigram_get_grammar_by_id (PROCESS_LM * *lm*, unsigned short *id*)

Find a grammar in LM by its ID number.

**Parameters:**

***lm*** [in] LM process instance

***id*** [in] ID number

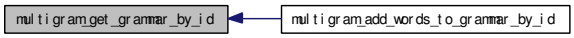
**Returns:**

poitner to the grammar, or NULL if not found.

Definition at line 1052 of file multi-gram.c.

Referenced by multigram_add_words_to_grammar_by_id().

Here is the caller graph for this function:




#### 6.7.2.18 boolean multigram_add_words_to_grammar (PROCESS_LM * *lm*, MULTIGRAM * *m*, WORD_INFO * *winfo*)

Append words to a grammar.

Category IDs of grammar in the adding words will be copied as is to the target grammar, so they should be set beforehand correctly. The whole tree lexicon will be rebuilt later.

Currently adding words to N-gram LM is not supported yet.

**Parameters:**

***lm*** [i/o] LM process instance

***m*** [i/o] grammar to which the winfo will be appended

***winfo*** [in] words to be added to the grammar

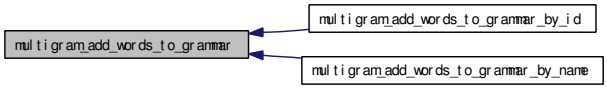
**Returns:**

TRUE on success, or FALSE on failure.

Definition at line 1101 of file multi-gram.c.

Referenced by multigram_add_words_to_grammar_by_id(), and multigram_add_words_to_grammar_-by_name().

Here is the caller graph for this function:



#### 6.7.2.19 boolean multigram_add_words_to_grammar_by_name (PROCESS_LM * *lm*, char * *gramname*, WORD_INFO * *winfo*)

Append words to a grammar, given by its name.

Call multigram_add_words_to_grammar() with target grammar specified by its name.

**Parameters:**

***lm*** [i/o] LM process instance

***gramname*** [in] name of the grammar to which the winfo will be appended

***winfo*** [in] words to be added to the grammar

**Returns:**

TRUE on success, or FALSE on failure.

Definition at line 1154 of file multi-gram.c.

#### 6.7.2.20 boolean multigram_add_words_to_grammar_by_id (PROCESS_LM * *lm*, unsigned short *id*, WORD_INFO * *winfo*)

Append words to a grammar, given by its ID number.

Call multigram_add_words_to_grammar() with target grammar specified by its number.

**Parameters:**

***lm*** [i/o] LM process instance

***id*** [in] ID number of the grammar to which the winfo will be appended

***winfo*** [in] words to be added to the grammar

**Returns:**

TRUE on success, or FALSE on failure.

Definition at line 1185 of file multi-gram.c.

## 6.8 Jconf configuration API

Collaboration diagram for Jconf configuration API:


Functions to load / create configuration parameters.

### Functions

- JCONF_AM * j_jconf_am_new ()
  *Allocate a new acoustic model (AM) parameter structure.*
- void j_jconf_am_free (JCONF_AM *amconf)
  *Release an acoustic model (AM) parameter structure Default parameter values are set to it.*
- boolean j_jconf_am_regist (Jconf *jconf, JCONF_AM *amconf, char *name)
  *Register AM configuration to global jconf.*
- JCONF_LM * j_jconf_lm_new ()
  *Allocate a new language model (LM) parameter structure.*
- void j_jconf_lm_free (JCONF_LM *lmconf)
  *Release a language model (LM) parameter structure.*
- boolean j_jconf_lm_regist (Jconf *jconf, JCONF_LM *lmconf, char *name)
  *Register LM configuration to global jconf.*
- JCONF_SEARCH * j_jconf_search_new ()
  *Allocate a new search (SEARCH) parameter structure.*
- void j_jconf_search_free (JCONF_SEARCH *sconf)
  *Release a search (SEARCH) parameter structure.*
- boolean j_jconf_search_regist (Jconf *jconf, JCONF_SEARCH *sconf, char *name)
  *Register SEARCH configuration to global jconf.*
- Jconf * j_jconf_new ()
  *Allocate a new global configuration parameter structure.*
- void j_jconf_free (Jconf *jconf)
  *Free a global configuration parameter structure.*
- int j_config_load_args (Jconf *jconf, int argc, char *argv[ ])
  *Load parameters from command argments, and set to each configuration instances in jconf.*
- int j_config_load_file (Jconf *jconf, char *filename)
  *Load parameters from a jconf file and set to each configuration instances in jconf.*

- Jconf * j_config_load_args_new (int argc, char *argv[ ])

  *Create a new configuration instance and load parameters from command argments.*

- Jconf * j_config_load_file_new (char *filename)

  *Create a new configuration instance and load parameters from a jconf file.*

- JCONF_AM * j_get_amconf_by_name (Jconf *jconf, char *name)

  *Get AM configuration structure in jconf by its name.*

- JCONF_AM * j_get_amconf_by_id (Jconf *jconf, int id)

  *Get AM configuration structure in jconf by its id.*

- JCONF_AM * j_get_amconf_default (Jconf *jconf)

  *Return default AM configuration.*

- JCONF_LM * j_get_lmconf_by_name (Jconf *jconf, char *name)

  *Get LM configuration structure in jconf by its name.*

- JCONF_LM * j_get_lmconf_by_id (Jconf *jconf, int id)

  *Get LM configuration structure in jconf by its id.*

- JCONF_SEARCH * j_get_searchconf_by_name (Jconf *jconf, char *name)

  *Get SEARCH configuration structure in jconf by its name.*

- JCONF_SEARCH * j_get_searchconf_by_id (Jconf *jconf, int id)

  *Get SEARCH configuration structure in jconf by its id.*

- boolean j_jconf_finalize (Jconf *jconf)

  *Check and finalize jconf parameters.*

### 6.8.1 Detailed Description

Functions to load / create configuration parameters.

### 6.8.2 Function Documentation

#### 6.8.2.1 JCONF_AM* j_jconf_am_new ()

Allocate a new acoustic model (AM) parameter structure.

Default parameter values are set to it.

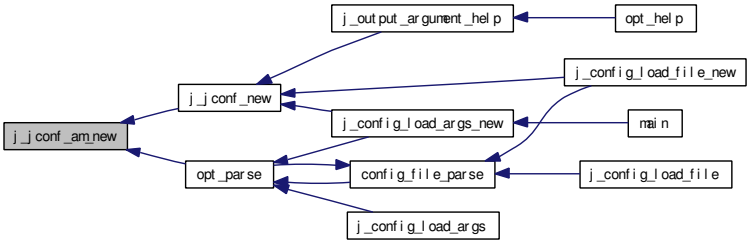
**Returns:**

the newly allocated AM parameter structure

Definition at line 336 of file instance.c.

Referenced by j_jconf_new(), and opt_parse().

Here is the caller graph for this function:




### 6.8.2.2 void j_jconf_am_free (JCONF_AM * *amconf*)

Release an acoustic model (AM) parameter structure Default parameter values are set to it.

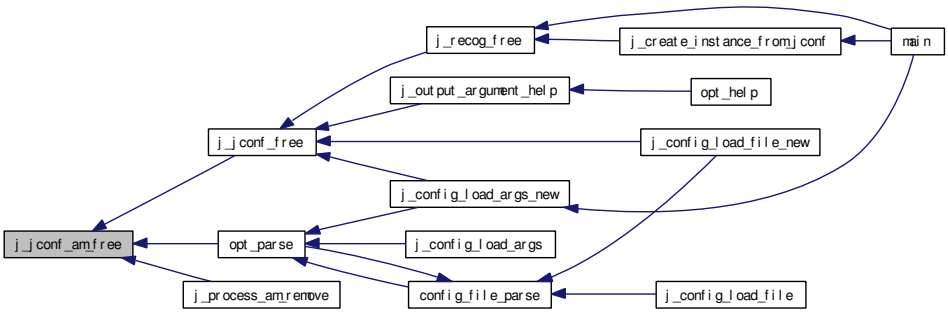
**Parameters:**

***amconf*** [in] AM configuration

Definition at line 363 of file instance.c.

Referenced by j_jconf_free(), j_process_am_remove(), and opt_parse().

Here is the caller graph for this function:




### 6.8.2.3 boolean j_jconf_am_regist (Jconf * *jconf*, JCONF_AM * *amconf*, char * *name*)

Register AM configuration to global jconf.

Returns error if the same name already exist in the jconf.

**Parameters:**

***jconf*** [i/o] global jconf

***amconf*** [in] AM configuration to register

***name*** [in] module name

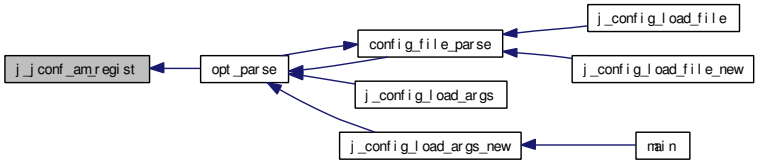
**Returns:**

TRUE on success, FALSE on failure

Definition at line 389 of file instance.c.

Referenced by opt_parse().

Here is the caller graph for this function:




#### 6.8.2.4 JCONF_LM* j_jconf_lm_new ()

Allocate a new language model (LM) parameter structure.

Default parameter values are set to it.

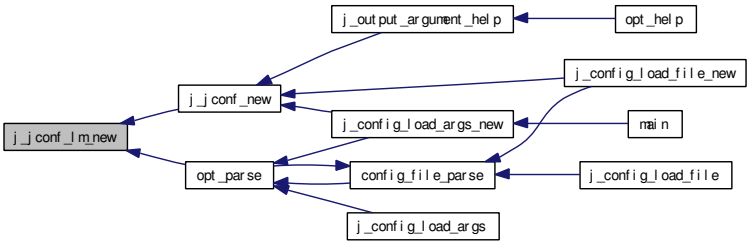
**Returns:**

the newly allocated LM parameter structure.

Definition at line 440 of file instance.c.

Referenced by j_jconf_new(), and opt_parse().

Here is the caller graph for this function:




#### 6.8.2.5 void j_jconf_lm_free (JCONF_LM * *lmconf*)

Release a language model (LM) parameter structure.

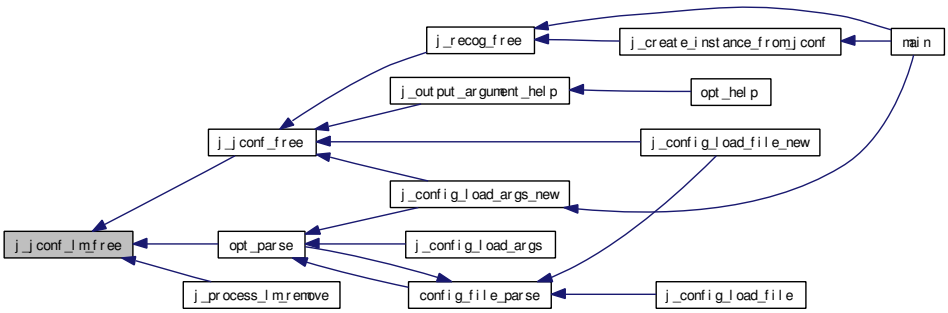
**Parameters:**

***lmconf*** [in] LM parameter structure

Definition at line 465 of file instance.c.

Referenced by j_jconf_free(), j_process_lm_remove(), and opt_parse().

Here is the caller graph for this function:




#### 6.8.2.6 boolean j_jconf_lm_regist (Jconf * *jconf*, JCONF_LM * *lmconf*, char * *name*)

Register LM configuration to global jconf.

Returns error if the same name already exist in the jconf.

**Parameters:**

***jconf*** [i/o] global jconf

***lmconf*** [in] LM configuration to register

***name*** [in] module name

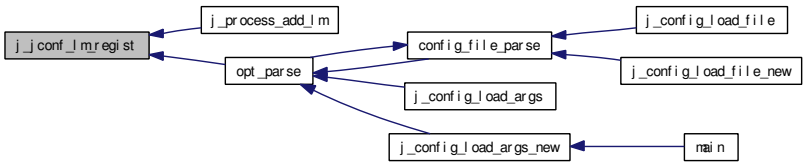
**Returns:**

TRUE on success, FALSE on failure

Definition at line 491 of file instance.c.

Referenced by j_process_add_lm(), and opt_parse().

Here is the caller graph for this function:




#### 6.8.2.7 JCONF_SEARCH* j_jconf_search_new ()

Allocate a new search (SEARCH) parameter structure.

Default parameter values are set to it.

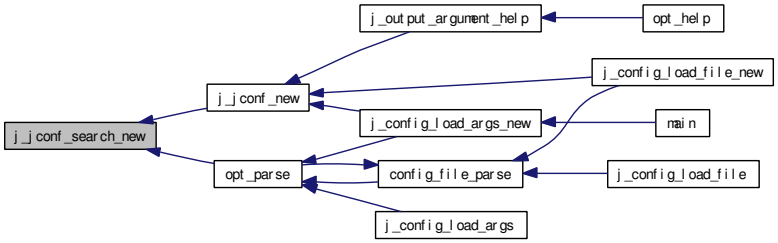
**Returns:**

the newly allocated SEARCH parameter structure.

Definition at line 542 of file instance.c.

Referenced by j_jconf_new(), and opt_parse().

Here is the caller graph for this function:




#### 6.8.2.8 void j_jconf_search_free (JCONF_SEARCH * *sconf*)

Release a search (SEARCH) parameter structure.

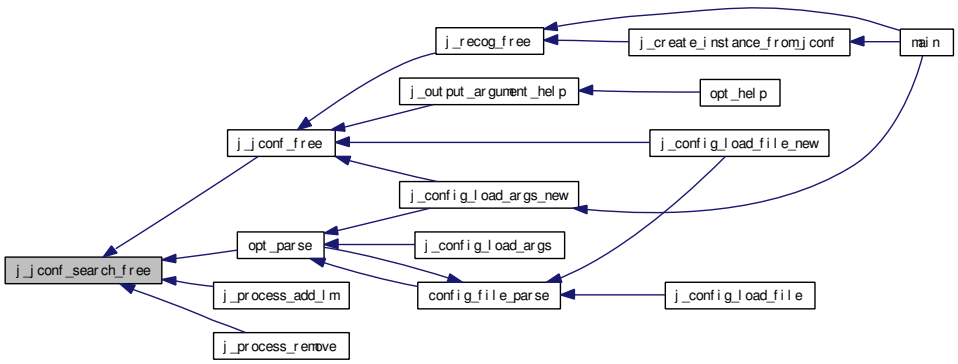
**Parameters:**

***sconf*** [in] SEARCH parameter structure

Definition at line 567 of file instance.c.

Referenced by j_jconf_free(), j_process_add_lm(), j_process_remove(), and opt_parse().

Here is the caller graph for this function:




#### 6.8.2.9 boolean j_jconf_search_regist (Jconf * *jconf*, JCONF_SEARCH * *sconf*, char * *name*)

Register SEARCH configuration to global jconf.

Returns error if the same name already exist in the jconf.

**Parameters:**

***jconf*** [i/o] global jconf

***sconf*** [in] SEARCH configuration to register

***name*** [in] module name

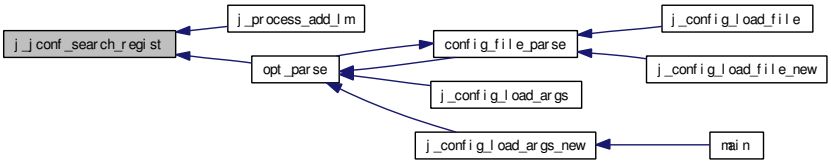
**Returns:**

TRUE on success, FALSE on failure

Definition at line 593 of file instance.c.

Referenced by j_process_add_lm(), and opt_parse().

Here is the caller graph for this function:




#### 6.8.2.10 Jconf* j_jconf_new ()

Allocate a new global configuration parameter structure.

JCONF_AM, JCONF_LM, JCONF_SEARCH are defined one for each, and assigned to the newly allocated structure as initial instances.

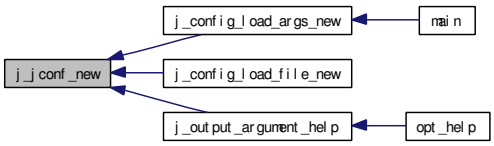
**Returns:**

the newly allocated global configuration parameter structure.

Definition at line 649 of file instance.c.

Referenced by j_config_load_args_new(), j_config_load_file_new(), and j_output_argument_help().

Here is the caller graph for this function:




#### 6.8.2.11 void j_jconf_free (Jconf * *jconf*)

Free a global configuration parameter structure.

All JCONF_AM, JCONF_LM, JCONF_SEARCH are also released.

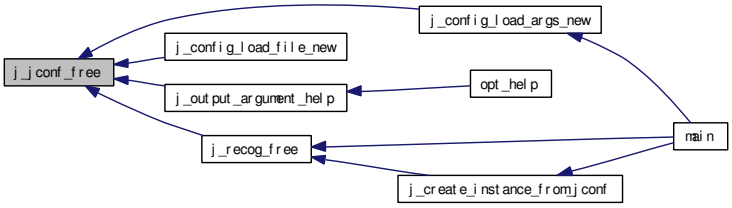
**Parameters:**

***jconf*** [in] global configuration parameter structure

Definition at line 702 of file instance.c.

Referenced by j_config_load_args_new(), j_config_load_file_new(), j_output_argument_help(), and j_recog_free().

Here is the caller graph for this function:




#### 6.8.2.12 int j_config_load_args (Jconf * *jconf*, int *argc*, char * *argv*[ ])

Load parameters from command argments, and set to each configuration instances in jconf.

**Parameters:**

***jconf*** [i/o] global configuration instance

***argc*** [in] number of arguments

***argv*** [in] list of argument strings

**Returns:**

0 on success, or -1 on failure.

Definition at line 319 of file jfunc.c.

#### 6.8.2.13 int j_config_load_file (Jconf * *jconf*, char * *filename*)

Load parameters from a jconf file and set to each configuration instances in jconf.

**Parameters:**

***jconf*** [i/o] glbal configuration instance

***filename*** [in] jconf filename

**Returns:**

0 on sucess, or -1 on failure.

Definition at line 368 of file jfunc.c.

#### 6.8.2.14 Jconf* j_config_load_args_new (int *argc*, char * *argv*[ ])

Create a new configuration instance and load parameters from command argments.

**Parameters:**

***argc*** [in] number of arguments
***argv*** [in] list of argument strings

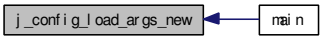
**Returns:**

the newly allocated global configuration instance.

Definition at line 417 of file jfunc.c.

Referenced by main().

Here is the caller graph for this function:




#### 6.8.2.15 Jconf* j_config_load_file_new (char * *filename*)

Create a new configuration instance and load parameters from a jconf file.

**Parameters:**

***filename*** [in] jconf filename

**Returns:**

the newly allocated global configuration instance.

Definition at line 468 of file jfunc.c.

#### 6.8.2.16 JCONF_AM* j_get_amconf_by_name (Jconf * *jconf*, char * *name*)

Get AM configuration structure in jconf by its name.

**Parameters:**

***jconf*** [in] global configuration
***name*** [in] AM module name

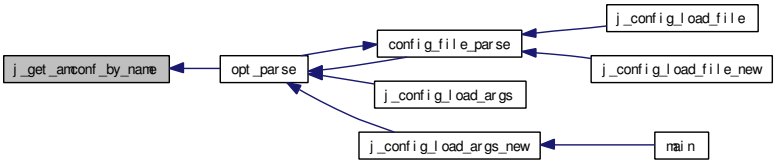
**Returns:**

the specified AM configuration, or NULL if not found.

Definition at line 773 of file jfunc.c.

Referenced by opt_parse().

Here is the caller graph for this function:



#### 6.8.2.17 JCONF_AM* j_get_amconf_by_id (Jconf * *jconf*, int *id*)

Get AM configuration structure in jconf by its id.

**Parameters:**

***jconf*** [in] global configuration

***id*** [in] AM module ID

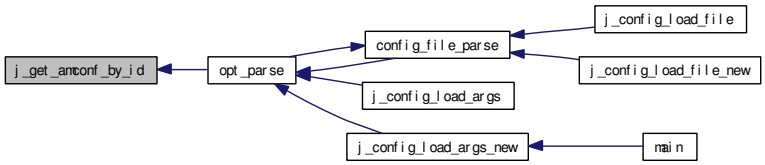
**Returns:**

the specified AM configuration, or NULL if not found.

Definition at line 807 of file jfunc.c.

Referenced by opt_parse().

Here is the caller graph for this function:




#### 6.8.2.18 JCONF_AM* j_get_amconf_default (Jconf * *jconf*)

Return default AM configuration.

If multiple AM configuration exists, return the latest one.

**Parameters:**

***jconf*** [in] global configuration

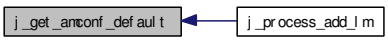
**Returns:**

the specified AM configuration, or NULL if not found.

Definition at line 844 of file jfunc.c.

Referenced by j_process_add_lm().

Here is the caller graph for this function:




#### 6.8.2.19 JCONF_LM* j_get_lmconf_by_name (Jconf * *jconf*, char * *name*)

Get LM configuration structure in jconf by its name.

**Parameters:**

***jconf*** [in] global configuration
***name*** [in] LM module name

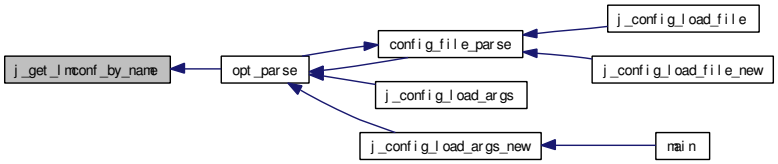
**Returns:**

the specified LM configuration, or NULL if not found.

Definition at line 871 of file jfunc.c.

Referenced by opt_parse().

Here is the caller graph for this function:




#### 6.8.2.20 JCONF_LM* j_get_lmconf_by_id (Jconf * *jconf*, int *id*)

Get LM configuration structure in jconf by its id.

**Parameters:**

***jconf*** [in] global configuration
***id*** [in] LM module ID

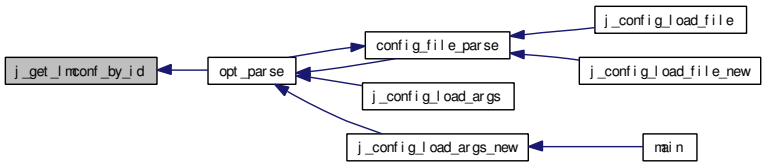
**Returns:**

the specified LM configuration, or NULL if not found.

Definition at line 905 of file jfunc.c.

Referenced by opt_parse().

Here is the caller graph for this function:




#### 6.8.2.21 JCONF_SEARCH* j_get_searchconf_by_name (Jconf * *jconf*, char * *name*)

Get SEARCH configuration structure in jconf by its name.

**Parameters:**

***jconf*** [in] global configuration

***name*** [in] SEARCH module name

**Returns:**

the found SEARCH configuration, or NULL if not found.

Definition at line 939 of file jfunc.c.

#### 6.8.2.22 JCONF_SEARCH* j_get_searchconf_by_id (Jconf * *jconf*, int *id*)

Get SEARCH configuration structure in jconf by its id.

**Parameters:**

***jconf*** [in] global configuration

***id*** [in] SEARCH module ID

**Returns:**

the found SEARCH configuration, or NULL if not found.

Definition at line 973 of file jfunc.c.

#### 6.8.2.23 boolean j_jconf_finalize (Jconf * *jconf*)

Check and finalize jconf parameters.

This functions parse through the global jconf configuration parameters. This function checks for value range of variables, file existence, competing specifications among variables or between variables and models, calculate some parameters from the given values, etc.

This function should be called just after all values are set by jconf, command argument or by user application, and before creating engine instance and loading models.

**Parameters:**

***jconf*** [i/o] global jconf configuration structure

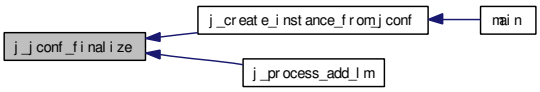
**Returns:**

TRUE when all check has been passed, or FALSE if not passed.

Definition at line 95 of file m_chkparam.c.

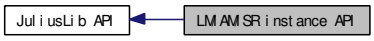
Referenced by j_create_instance_from_jconf(), and j_process_add_lm().

Here is the caller graph for this function:



## 6.9 LM/AM/SR instance API

Collaboration diagram for LM/AM/SR instance API:




Functions to handle modules and processes directly.

### Functions

- Recog * j_recog_new ()
  *Allocate memory for a new engine instance.*
- void j_recog_free (Recog *recog)
  *Free an engine instance.*
- Recog * j_create_instance_from_jconf (Jconf *jconf)
  *Instanciate / generate a new engine instance according to the given global configuration instance.*
- boolean j_load_am (Recog *recog, JCONF_AM *amconf)
  *Load an acoustic model.*
- boolean j_load_lm (Recog *recog, JCONF_LM *lmconf)
  *Load a language model.*
- boolean j_load_all (Recog *recog, Jconf *jconf)
  *Read in all models for recognition.*
- boolean j_launch_recognition_instance (Recog *recog, JCONF_SEARCH *sconf)
  *Launch a recognition process instance.*
- boolean j_final_fusion (Recog *recog)
  *Combine all loaded models and settings into one engine instance.*

### 6.9.1 Detailed Description

Functions to handle modules and processes directly.

### 6.9.2 Function Documentation

#### 6.9.2.1 Recog* j_recog_new ()

Allocate memory for a new engine instance.

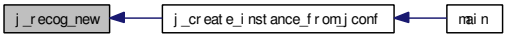
**Returns:**

the newly allocated engine instance.

Definition at line 746 of file instance.c.

Referenced by j_create_instance_from_jconf().

Here is the caller graph for this function:




#### 6.9.2.2 void j_recog_free (Recog * *recog*)

Free an engine instance.

All allocated memories in the instance will be also released.

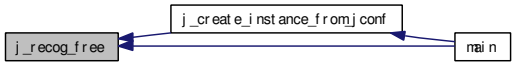
**Parameters:**

***recog*** [in] engine instance.

Definition at line 800 of file instance.c.

Referenced by j_create_instance_from_jconf(), and main().

Here is the caller graph for this function:




#### 6.9.2.3 Recog* j_create_instance_from_jconf (Jconf * *jconf*)

Instanciate / generate a new engine instance according to the given global configuration instance.

It inspects all parameters in the global configuration instance, load all models into memory, build tree lexicons, allocate work area and caches. It does all setup to start recognition except A/D-in initialization.

**Parameters:**

***jconf*** [in] gloabl configuration instance

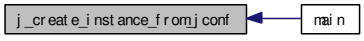
**Returns:**

the newly created engine instance.

Definition at line 661 of file jfunc.c.

Referenced by main().

Here is the caller graph for this function:



#### 6.9.2.4 boolean j_load_am (Recog * *recog*, JCONF_AM * *amconf*)

Load an acoustic model.

This function will create an AM process instance using the given AM configuration, and load models specified in the configuration into the instance. Then the created instance will be installed to the engine instance. The amconf should be registered to the global jconf before calling this function.

**Parameters:**

***recog*** [i/o] engine instance

***amconf*** [in] AM configuration to load

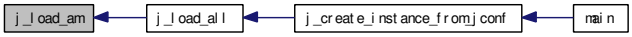
**Returns:**

TRUE on success, or FALSE on error.

Definition at line 428 of file m_fusion.c.

Referenced by j_load_all().

Here is the caller graph for this function:




#### 6.9.2.5 boolean j_load_lm (Recog * *recog*, JCONF_LM * *lmconf*)

Load a language model.

This function will create an LM process instance using the given LM configuration, and load models specified in the configuration into the instance. Then the created instance will be installed to the engine instance. The lmconf should be registered to the recog->jconf before calling this function.

To convert phoneme sequence to triphone at loading, you should specify which AM to use with this LM by the argument am.

**Parameters:**

***recog*** [i/o] engine instance

***lmconf*** [in] LM configuration to load

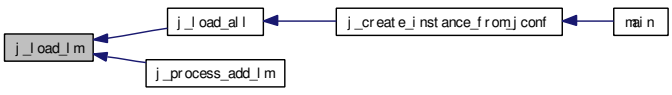
**Returns:**

TRUE on success, or FALSE on error.

Definition at line 517 of file m_fusion.c.

Referenced by j_load_all(), and j_process_add_lm().

Here is the caller graph for this function:



#### 6.9.2.6 boolean j_load_all (Recog * *recog*, Jconf * *jconf*)

Read in all models for recognition.

This function create AM/LM processing instance for each AM/LM configurations in jconf. Then the model for each instance will be loaded into memory and set up for recognition. GMM will also be read here.

**Parameters:**

***recog*** [i/o] engine instance

***jconf*** [in] global configuration variables

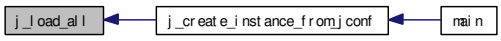
**Returns:**

TRUE on success, FALSE on failure.

Definition at line 615 of file m_fusion.c.

Referenced by j_create_instance_from_jconf().

Here is the caller graph for this function:




#### 6.9.2.7 boolean j_launch_recognition_instance (Recog * *recog*, JCONF_SEARCH * *sconf*)

Launch a recognition process instance.

This function will create an recognition process instance using the given SEARCH configuration, and launch recognizer for the search. Then the created instance will be installed to the engine instance. The sconf should be registered to the global jconf before calling this function.

**Parameters:**

***recog*** [i/o] engine instance

***sconf*** [in] SEARCH configuration to launch

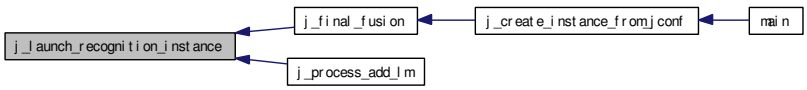
**Returns:**

TRUE on success, or FALSE on error.

Definition at line 865 of file m_fusion.c.

Referenced by j_final_fusion(), and j_process_add_lm().

Here is the caller graph for this function:



#### 6.9.2.8 boolean j_final_fusion (Recog * *recog*)

Combine all loaded models and settings into one engine instance.

This function will finalize preparation of recognition:

- create required MFCC calculation instances,
- create recognition process instance for specified LM/AM combination,
- set model-specific recognition parameters,
- build tree lexicon for each process instance for the 1st pass,
- prepare work area and cache area for recognition,
- initialize some values / work area for frontend processing.

After this function, all recognition setup was done and we are ready for start recognition.

This should be called after j_jconf_finalize() and j_load_all() has been completed. You should put the jconf at recog->jconf before calling this function.

**Parameters:**

***recog*** [in] engine instance

**Returns:**

TRUE when all initialization successfully done, or FALSE if any error has been occured.

Definition at line 1111 of file m_fusion.c.

Referenced by j_create_instance_from_jconf().

Here is the caller graph for this function: